Performanse računarskih sistema

Projekat

Uroš Lončar 0691/19

1. jul 2023.

Projekat je odrađen u *Python* programskom jeziku uz korišćenje biblioteka *numpy* (za računanje i generisanje poslova sa eksponencijalnom raspodelom), *multiprocessing* (za paralelno izvršavanje većeg broja simulacija), *pprint* (za generisanje citljivih teksutalnih dokumenata), *matplotlib* (za iscrtavanje grafika) i *openpyxl* (za generisanje Excel dokumenata).

1 Analitičko rešavanje

Analitičko rešavanje se deli u više sekvencijalnih koraka koji koriste rezultate prethodnog koraka. Svi koraci su ponaosob ponovljeni za svako traženo K (broj korisničkih diskova, 2 do 5).

1.1 Postavka zadatka

Iz postavke zadatka možemo da izvučemo matricu verovatnoća prelaza sa jednog servera na drugi i vektor performansi.

Matricu verovatnoće prelaza i vektor performansi ćemo morati dinamički da menjamo u odnosu na broj korisničkih diskova za koje računamo.

1.2 Određivanje odnosa protoka kroz server i jačine ulaznog toka

Za ovaj korak će nam trebati matrica verovatnoće prelaza P. Možemo doći do odnosa protoka kroz server i jačine ulaznog toka primenjivanjem matrične metode za rešavanje otvorenih mreža korišćenjem formule $\lambda = (I - P^T)^{-1} \cdot \alpha$.

1.3 Određivanje granične vrednost i kritičnog resursa

Za ovaj korak će nam trebati prethodno izračunati odnosi protoka kroz server i jačinu ulaznog toka i vektor performansi. Sada želimo da izračunamo iskorišćenja U svih servera. Onaj server sa najvećim iskorišćenjem predstavlja kritični resurs našeg sistema i kako bi se uspostavio stacionarni režim mi moramo da ograničimo naš protok na protok tog servera. Na ovaj način smo našli maksimalno a za koje naš sistem radi.

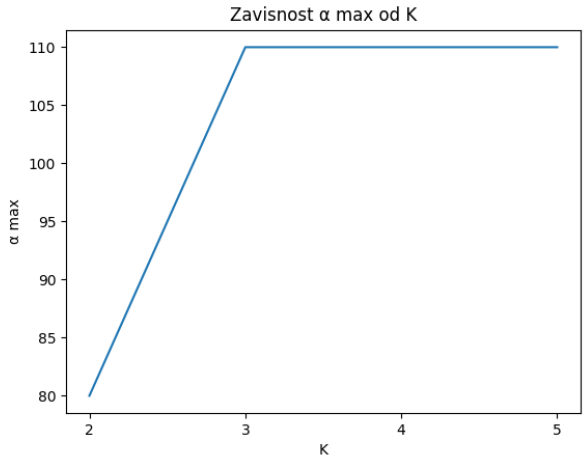
1.4 Određivanje traženih parametara

Preostaje nam da odredimo ostale tražene parametre; protok kroz sve resurse, iskorišćenje svih resursa, prosečan broj poslova na svakom resursu i vreme odziva celog sistema.

Ove parametre ćemo odrediti za svaku modifikaciju maksimalnog intenziteta ulaznog toka datih u zadatku (0.25, 0.50, 0.77, 0.99).

Kako smo u prethdnom korako našli maksimalan intenzitet ulaznog toka α sada možemo odrediti tačne protoke X pomoću izračunatih odnosa protoka i jačine ulaznog roka iz druge taćke. Množenjem protoka i vektora performansi određujemo iskorišćenost U. Iz iskorišćenosti možemo direktno odrediti srednji broj poslova na svakom resursu J. I vreme odziva svake komponente T i celog sistema *Toverall* možemo odrediti deljenjem srednjeg broja poslova na svakom resursu sa protokom.

Ovim smo odredili sve tražene parametre sistema.



Slika 1: Zavistnost maksimalne jačine ulaznog toka α od broja korisničkih diskova K

2 Simulacija

Simulaciju vršimo tako što pomoću klasa Active koja predstavlja jedan Server i klase Spreader koja je pomoćna klasa za nasumičnu distribuciju poslova kroz sistemi pravimo mrežu instanci tih klasa koja odgovara sistemu opisanom u zadatku.

U napravljenu mrežu onda šaljemo listu brojeva koji predstavljaju vreme ulaska svakog posla u sistemi. Vremena ulaska poslova u sistem su generisana sa eksponencijalnom raspodelom.

2.1 Klasa Active

Klasa `Active` predstavlja aktivnu serversku komponentu koja procesira poslove. Ima metodu `tick` koja procesira sledeći posao u redu, metodu `get_next_event_time` koja vraća vreme sledećeg događaja, i metodu `__init__` koja inicijalizuje aktivnu komponentu sa zadatim imenom i vremenom procesiranja jednog posla. Takođe ima metode za postavljanje ciljne komponente, dodavanje posla u red, i procesiranje posla.

U ovoj klasi se nalaze brojači koji će nam koristiti pri određivanju traženih parametara, konkretno waitingTime za srednji broj poslova na serveru, processedCount za protok i processedTime za iskorišćenost i srednji broj poslova na serveru.

Pri obradi posla ova klasa poslu dodaje vreme izračunato eksponencijalnom distribucijom kako bi simulirali obradu jednog posla.

Listing 1: Active class

```
class Active(Component):

        def tick(self):
                job = self.queue.popleft()

                self.waiting_time += max(self.cooldown - job.current_time, 0)

                job.current_time = max(job.current_time, self.cooldown)

                self.process(job)
```

```
def get_next_event_time(self):
        if not self.queue:
                return float("inf")

        return max(self.queue[0].current_time, self.cooldown)


def __init__(self, name: str, service_time: int):
        super().__init__(name)

        self.queue: Deque[Job] = deque()
        self._service_time = service_time
        self._target = None

        self.cooldown = 0

        self.waiting_time = 0
        self.processed_time = 0
        self.processed_count = 0


def setTarget(self, target: Component):
        self._target = target


def add(self, job: Job):
        self.queue.append(job)


def process(self, job: Job):
        added_time = np.random.exponential(scale=self._service_time) if self.

        self.processed_time += added_time
        self.processed_count += 1

        job.current_time += added_time

        self.cooldown = job.current_time

        if self._target:
                self._target.add(job)
```

2.2 Klasa Spreader

Klasa `Spreader` predstavlja širilicu koja distribuira poslove ka različitim ciljevima na osnovu težina. Ima metodu `tick` koja procesira sledeći posao u redu, metodu `get_next_event_time` koja vraća vreme sledećeg događaja, i metodu `__init__` koja inicijalizuje širilicu sa zadatim imenom. Takođe ima metode za dodavanje ciljne komponente sa odgovarajućom težinom, dodavanje posla u red, i procesiranje posla.

Listing 2: Spreader class

```
class Spreader(Component):
        def tick(self):
                job = self.queue.popleft()
                self.process(job)


        def get_next_event_time(self):
                if not self.queue:
                        return float("inf")

                return self.queue[0].current_time


        def __init__(self, name: str):
                super().__init__(name)

                self._targets = []
                self._weights = []

                self.queue: Deque[Job] = deque()


        def addTarget(self, target: Component, weight: int):
                self._targets.append(target)
                self._weights.append(weight)


        def add(self, job: Job):
                self.queue.append(job)


        def process(self, job: Job):
                target: Component = random.choices(
                                self._targets,
                                weights=self._weights
                        )[0]

                target.add(job)
```

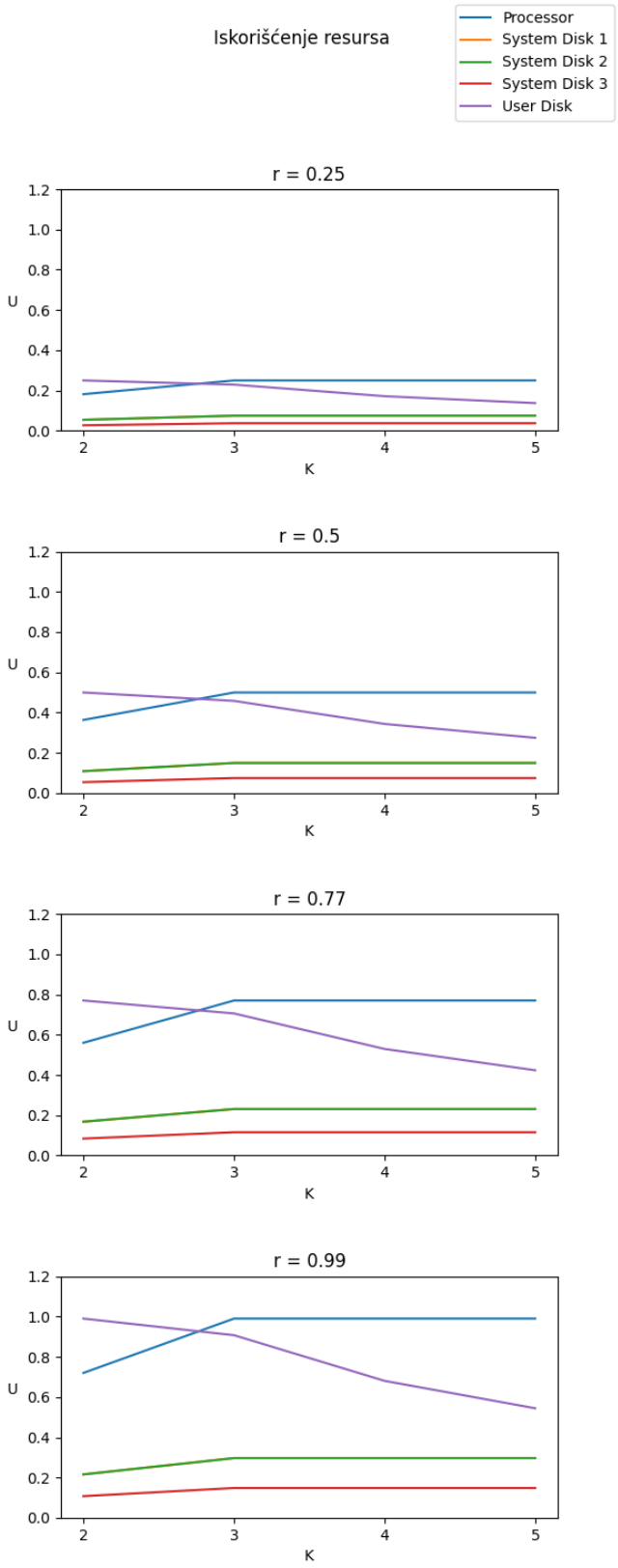
3 Analiza i dokumentovanje rezultata

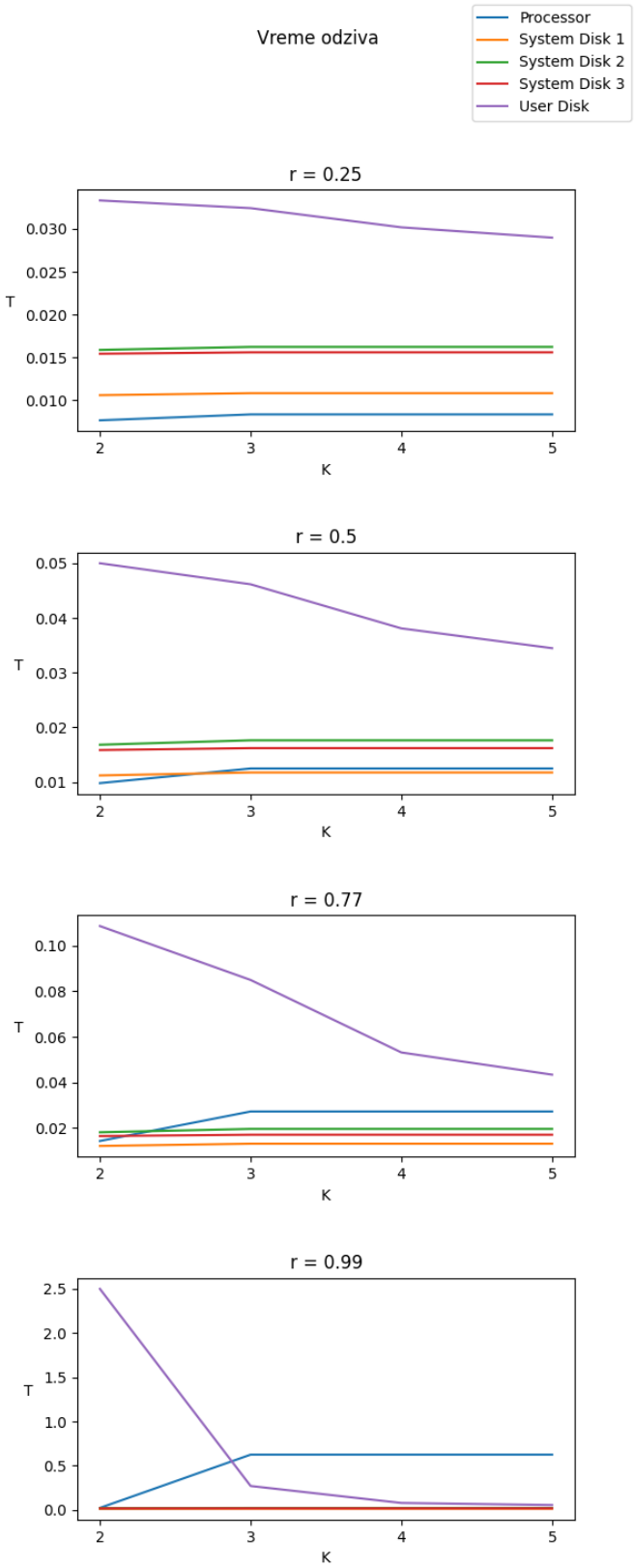
Sa generisanih grafikona iskorišćenosti vidimo da su za sistem sa dva korisnička diska baš oni kritični resursi jer jednostavno ne stižu dovoljno brzo da procesiraju poslove. Oni su takođe najsporija komponenta sistema.

Za tri i više korisničkih diskova kritični resurs postaje procesor.

Na grafikonima sa vremenom odziva sistema jasno vidimo da sa povećanjem broja korisničkih diskova, nama se smanjuje vreme odziva.

Pri posmatranju jedne simulacije imamo ne toliko preterana odstupanja od vrednosti izračunate analitičkim postupkom. Za kraća vremena simulacije možemo da vidimo da parametar za srednji broj poslova u resursu značajno odstupa od onoga što očekujemo. Ovo rešavamo povećavanjem vremena simulacije i/ili broja simulacija kako bi se nasumični parametri sistema uravnotežili.


Iskorišćenje resursa
Processor
System Disk 1
System Disk 2
System Disk 3
User Disk
r = 0.25
r = 0.5
r = 0.77
r = 0.99
U
K
0.0
0.2
0.4
0.6
0.8
1.0
1.2
2
3
4
5


Vreme odziva
Processor
System Disk 1
System Disk 2
System Disk 3
User Disk
r = 0.25
r = 0.5
r = 0.77
r = 0.99
T
K

Vreme odziva sistema